

COMPTE RENDU

TP1 JEE : Inversion de contrôle et Injection des dépendances

Par : Assimi DIALLO

Encadrant : M. YOUSSEFI

Ceci est le premier TP du module Architectures Distribuées JEE et Middlewares. Il nous permettra de faire une prise de contact avec l'inversion de contrôle et l'injection des dépendances et ça sera aussi l'occasion pour une première utilisation du framework Spring.

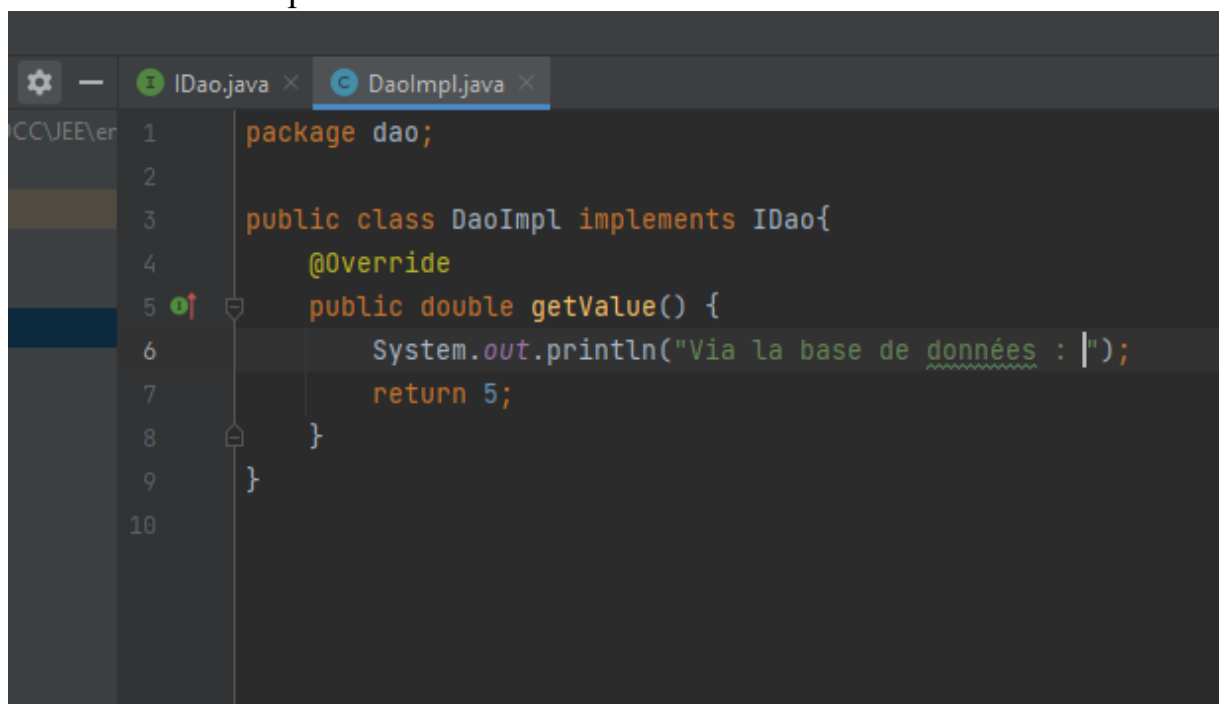
1- Création de l'interface IDao



The screenshot shows an IDE window with a single tab titled 'IDao.java'. The code defines a package 'dao' and a public interface 'IDao' with a method 'public double getValue()'. The file path on the left is 'CC\JEE\er'.

```
1 package dao;
2
3 public interface IDao {
4     public double getValue();
5 }
6
```

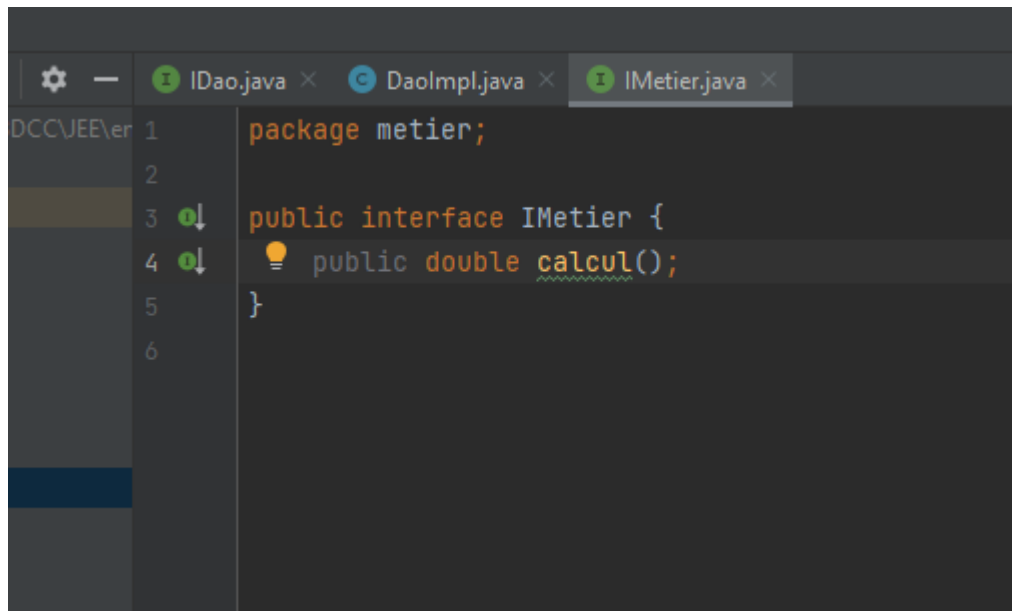
2- Création d'une implémentation de l'interface IDao



The screenshot shows an IDE window with two tabs: 'IDao.java' and 'DaoImpl.java'. The 'DaoImpl.java' tab is active, showing a class 'DaoImpl' that implements 'IDao'. The class has an '@Override' annotation and a 'public double getValue()' method that prints 'Via la base de données : |' and returns 5. The file path on the left is 'CC\JEE\er'.

```
1 package dao;
2
3 public class DaoImpl implements IDao{
4     @Override
5     public double getValue() {
6         System.out.println("Via la base de données : |");
7         return 5;
8     }
9 }
10
```

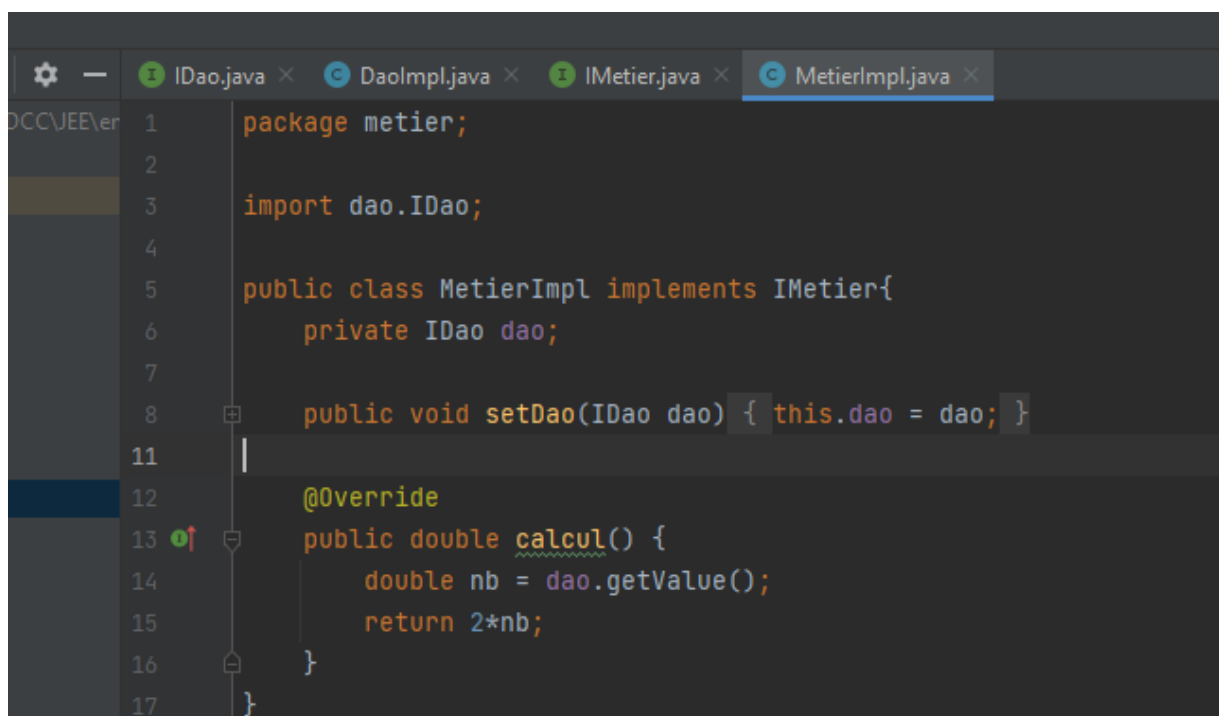
3- Création de l'interface IMetier



The screenshot shows an IDE with three tabs: IDao.java, DaoImpl.java, and IMetier.java. The IMetier.java tab is active, displaying the following code:

```
1 package metier;
2
3 public interface IMetier {
4     public double calcul();
5 }
6
```

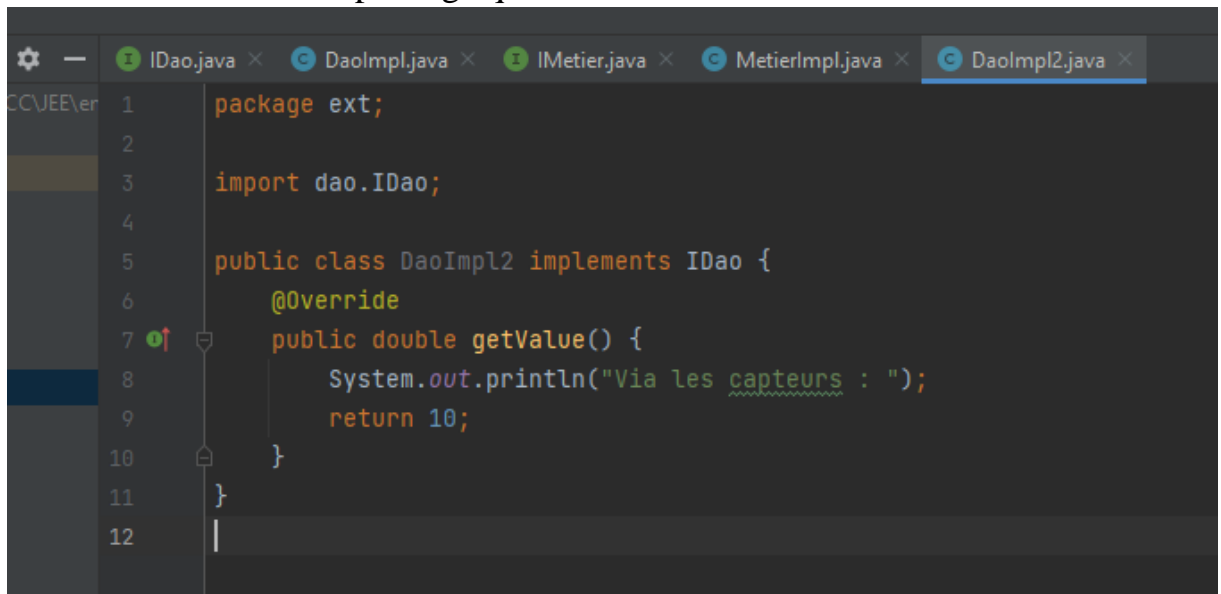
4- Création d'une implémentation de cette interface en utilisant le couplage faible



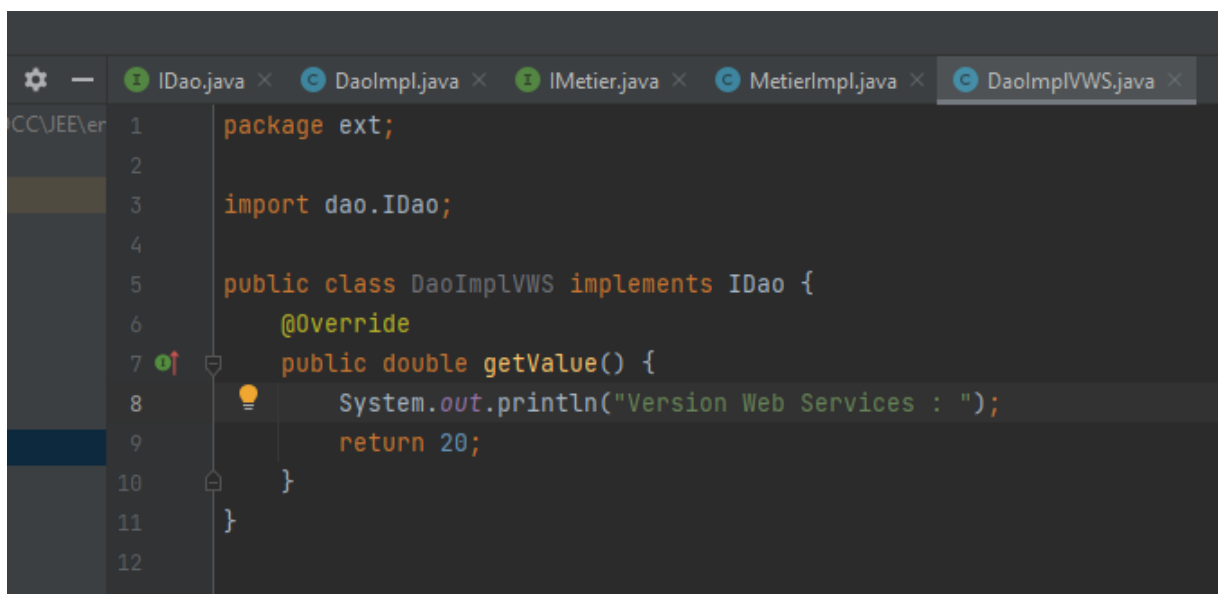
The screenshot shows an IDE with four tabs: IDao.java, DaoImpl.java, IMetier.java, and MetierImpl.java. The MetierImpl.java tab is active, displaying the following code:

```
1 package metier;
2
3 import dao.IDao;
4
5 public class MetierImpl implements IMetier{
6     private IDao dao;
7
8     public void setDao(IDao dao) { this.dao = dao; }
9
10
11
12     @Override
13     public double calcul() {
14         double nb = dao.getValue();
15         return 2*nb;
16     }
17 }
```

Nous allons créer 2 autres implémentations de l'interface IDao. Nous les mettrons dans un autre package que nous allons nommer **ext**.



```
1 package ext;
2
3 import dao.IDao;
4
5 public class DaoImpl2 implements IDao {
6     @Override
7     public double getValue() {
8         System.out.println("Via les capteurs : ");
9         return 10;
10    }
11 }
12
```



```
1 package ext;
2
3 import dao.IDao;
4
5 public class DaoImplVWS implements IDao {
6     @Override
7     public double getValue() {
8         System.out.println("Version Web Services : ");
9         return 20;
10    }
11 }
12
```

5- Faisons l'injection des dépendances :

a- Par instanciation statique

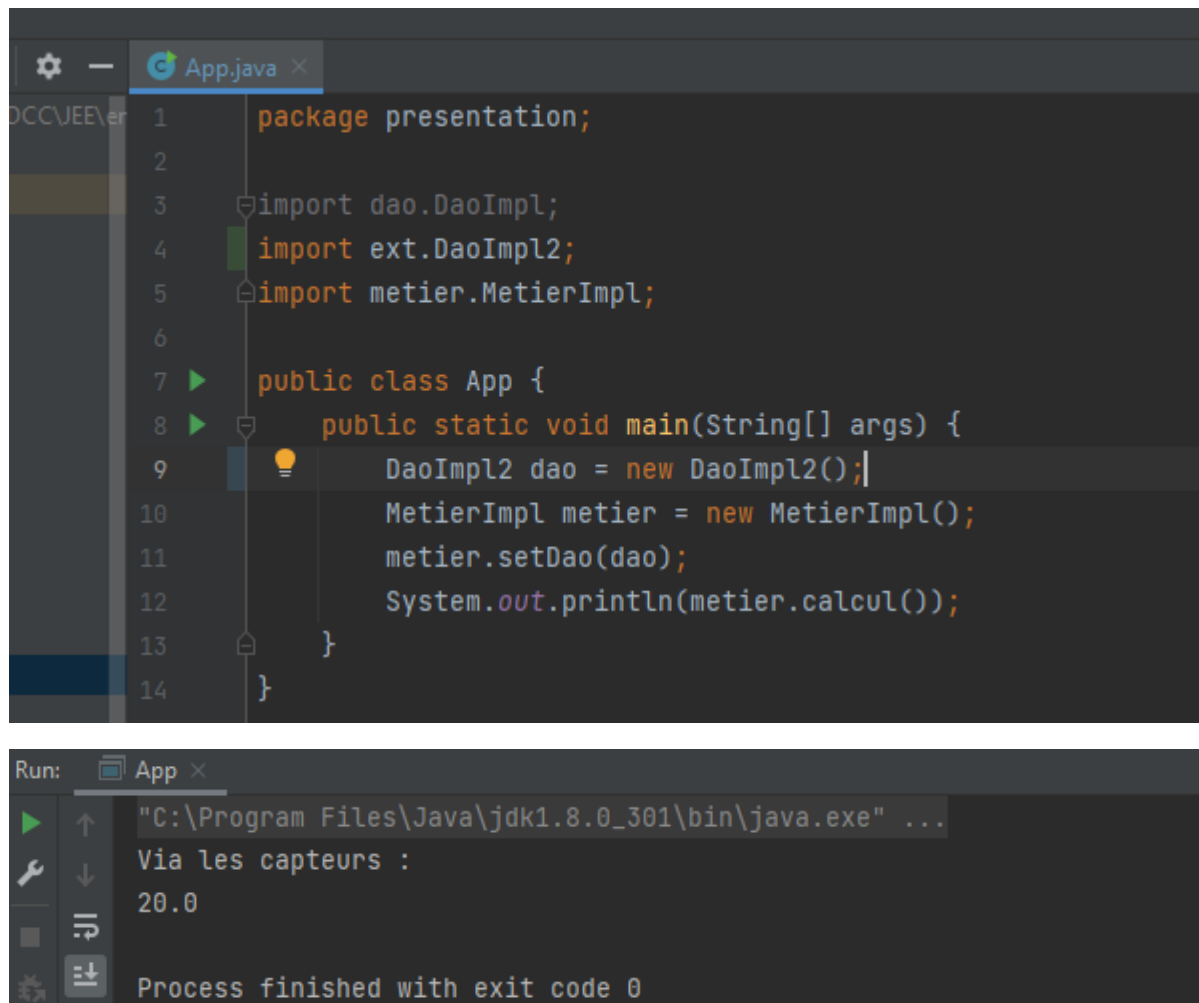
Pour cela nous allons créer une classe **App** dans un package **Présentation**, classe dans laquelle nous allons utiliser la méthode **main**.

```
1 package presentation;
2
3 import dao.DaoImpl;
4 import metier.MetierImpl;
5
6 public class App {
7     public static void main(String[] args) {
8         DaoImpl dao = new DaoImpl();
9         MetierImpl metier = new MetierImpl();
10        metier.setDao(dao);
11        System.out.println(metier.calcul());
12    }
13 }
```

```
Run: App x
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Via la base de données :
10.0
Process finished with exit code 0
```

Ici si nous choisissons d'utiliser une des versions, des implémentations de l'interface IDao, nous devons modifier la ligne 8. Le code n'est donc pas fermé à la modification bien qu'ouvert à l'extension.

Comme on peut le voir ci-dessous :



The image shows a screenshot of an IDE with two panels. The top panel displays the source code of a Java file named `App.java`. The code is as follows:

```
1 package presentation;
2
3 import dao.DaoImpl;
4 import ext.DaoImpl2;
5 import metier.MetierImpl;
6
7 public class App {
8     public static void main(String[] args) {
9         DaoImpl2 dao = new DaoImpl2();
10        MetierImpl metier = new MetierImpl();
11        metier.setDao(dao);
12        System.out.println(metier.calcul());
13    }
14 }
```

The bottom panel shows the output of running the application. The command executed is `"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...`. The output is:

```
Via les capteurs :
20.0
Process finished with exit code 0
```

b- Par instanciation dynamique

Nous allons donc passer par une instanciation dynamique pour rendre le code fermé à la modification. Pour cela nous allons créer un fichier de configuration dans lequel nous pourrions préciser l'implémentation IDao que nous souhaitons utiliser. Nous allons définir une classe **App2** dans laquelle nous allons utiliser la méthode **main**.

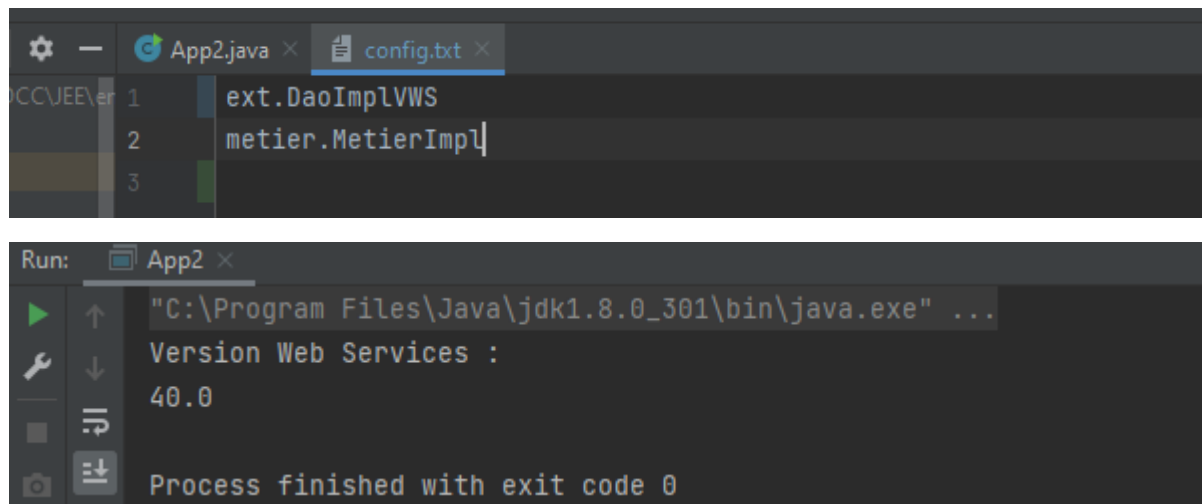
```
App2.java
4      import metier.IMetier;
5
6      import java.io.File;
7      import java.lang.reflect.Method;
8      import java.util.Scanner;
9
10     public class App2 {
11     public static void main(String[] args) {
12         try {
13             Scanner scanner= new Scanner( new File( pathname: "config.txt"));
14
15             String daoClassname = scanner.nextLine();
16             Class cdao = Class.forName(daoClassname);
17             IDao dao = (IDao) cdao.newInstance();
18
19             String metierClassname = scanner.nextLine();
20             Class cmetier = Class.forName(metierClassname);
21             IMetier metier = (IMetier) cmetier.newInstance();
22
23             Method meth = cmetier.getMethod( name: "setDao", IDao.class);
24             meth.invoke(metier,dao);
25             System.out.println(metier.calcul());
26         }catch (Exception e){
27             e.printStackTrace();
28         }
29     }
30 }
```

```
config.txt
1      dao.DaoImpl
2      metier.MetierImpl
```

Il s'affichera :

```
Run: App2
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Via la base de données :
10.0
Process finished with exit code 0
```

Modifier le fichier de configuration **config.txt** pour utiliser la version Services Web :



The image shows two screenshots from an IDE. The top screenshot shows a file named `config.txt` with the following content:

```
1 ext.DaoImplVWS
2 metier.MetierImpl
3
```

The bottom screenshot shows the `Run` console for `App2`. It displays the command executed:

```
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
```

The output shows:

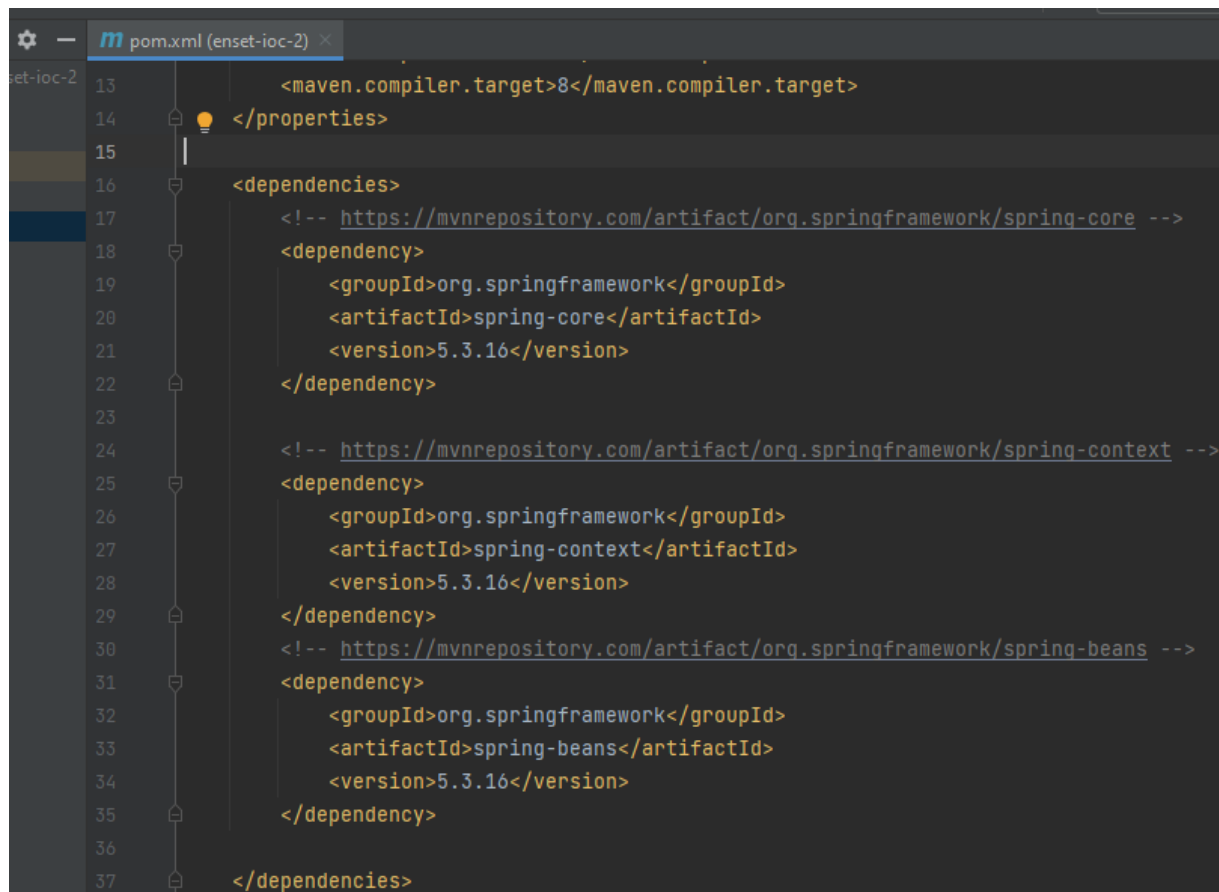
```
Version Web Services :
40.0
```

At the bottom, it states: `Process finished with exit code 0`.

Nous venons ainsi de faire l'injection des dépendances par instanciation statique puis par instanciation dynamique.

c- En utilisant le Framework Spring

Ici nous allons créer un nouveau projet avec les mêmes packages, classes et interfaces que le précédent projet. La différence est juste qu'ici nous allons créer un projet **Maven** pour pouvoir ajouter les dépendances nécessaires à l'utilisation du framework **Spring**. Ces dépendances sont ajoutées dans le fichier `pom.xml`.

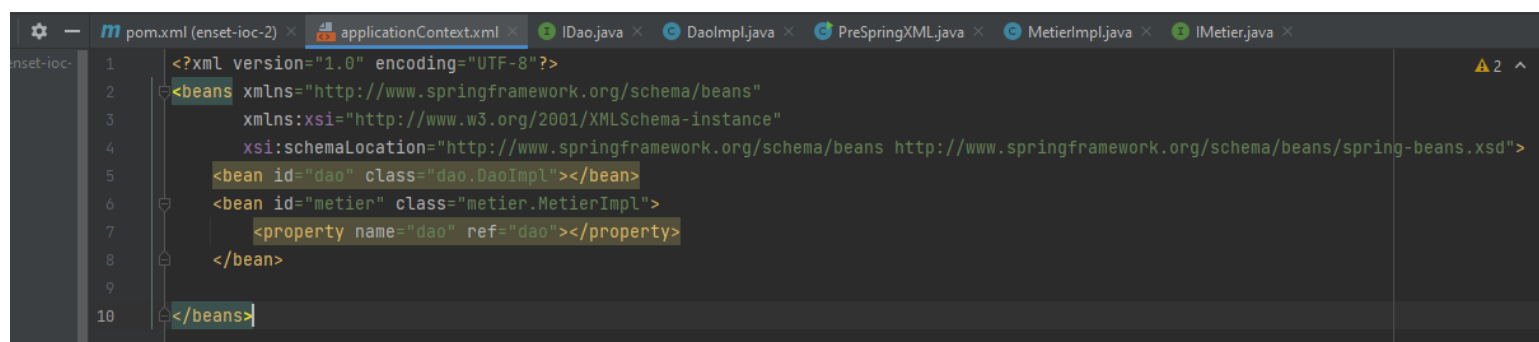


```
13      <maven.compiler.target>8</maven.compiler.target>
14    </properties>
15
16    <dependencies>
17      <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
18      <dependency>
19        <groupId>org.springframework</groupId>
20        <artifactId>spring-core</artifactId>
21        <version>5.3.16</version>
22      </dependency>
23
24      <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
25      <dependency>
26        <groupId>org.springframework</groupId>
27        <artifactId>spring-context</artifactId>
28        <version>5.3.16</version>
29      </dependency>
30      <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
31      <dependency>
32        <groupId>org.springframework</groupId>
33        <artifactId>spring-beans</artifactId>
34        <version>5.3.16</version>
35      </dependency>
36
37    </dependencies>
```

Nous allons faire l'injection des dépendances de 2 façons :

- Version XML

On crée un fichier XML (Spring Config) dans le dossier **resources** et on aura dans le fichier :



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
5   <bean id="dao" class="dao.DaoImpl"></bean>
6   <bean id="metier" class="metier.MetierImpl">
7     <property name="dao" ref="dao"></property>
8   </bean>
9
10 </beans>
```

Dans une classe **PreSpringXML** qui contient une méthode **main**, on aura

```
et-ioc- pom.xml (enset-ioc-2) x applicationContext.xml x PreSpringXML.java x
1 package presentation;
2
3 import metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class PreSpringXML {
8
9     public static void main(String[] args) {
10         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
11         IMetier metier = (IMetier) context.getBean("metier");
12         System.out.println(metier.calcul());
13     }
14 }
15
```

Si on exécute on obtient :

```
Run: PreSpringXML x
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Via la base de données :
10.0
Process finished with exit code 0
```

Pour utiliser une autre version, on modifie dans le fichier XML :

```
et-ioc- pom.xml (enset-ioc-2) x applicationContext.xml x PreSpringXML.java x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5       <bean id="dao" class="ext.DaoImplVWS"></bean>
6       <bean id="metier" class="metier.MetierImpl">
7         <property name="dao" ref="dao"></property>
8       </bean>
9
10 </beans>
```

Quand on exécute après cette modification :

```
Run: PreSpringXML x
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Version Web Services :
40.0
Process finished with exit code 0
```

- Version annotations

Ici on ajoute l'attribut `@Component` dans les implémentations de l'interface IDao, comme suit :

```
DaoImpl.java x
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("dao")
6 public class DaoImpl implements IDao{
7     @Override
8     public double getValue() {
9         System.out.println("Via la base de données : ");
10        return 5;
11    }
12 }
```

```
1 package ext;
2
3 import dao.IDao;
4 import org.springframework.stereotype.Component;
5
6 @Component("dao2")
7 public class DaoImpl2 implements IDao {
8     @Override
9     public double getValue() {
10         System.out.println("Via les capteurs : ");
11         return 10;
12     }
13 }
```

```
1 package ext;
2
3 import dao.IDao;
4 import org.springframework.stereotype.Component;
5
6 @Component("daoVWS")
7 public class DaoImplVWS implements IDao {
8     @Override
9     public double getValue() {
10         System.out.println("Version Web Services : ");
11         return 20;
12     }
13 }
```

Dans l'implémentation MetierImpl de l'interface on aura :

```
MetierImpl.java
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.beans.factory.annotation.Qualifier;
6  import org.springframework.stereotype.Component;
7
8  @Component
9  public class MetierImpl implements IMetier{
10     @Autowired
11     @Qualifier("dao")
12     private IDao dao;
13
14     public void setDao(IDao dao) { this.dao = dao; }
15
16
17
18     @Override
19     public double calcul() {
20         double nb = dao.getValue();
21         return 2*nb;
22     }
23 }
```

Dans l'attribut `@Qualifier` on précise l'implémentation `IDao` que nous souhaitons utiliser. Ici on utilise la version base de données. On va donc créer une classe **PresAnnot** va nous permettre d'exécuter notre programme.

```
PresAnnot.java
1  package presentation;
2
3  import metier.IMetier;
4  import org.springframework.context.ApplicationContext;
5  import org.springframework.context.annotation.AnnotationConfigApplicationContext;
6
7  public class PresAnnot {
8      public static void main(String[] args) {
9          ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "dao","metier","ext");
10         IMetier metier = context.getBean(IMetier.class);
11         System.out.println(metier.calcul());
12     }
13 }
14 |
```

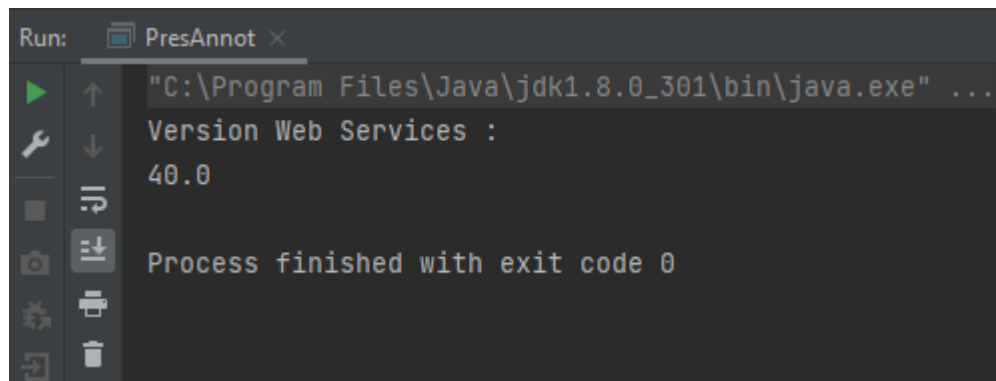
Quand on exécute le programme, on a :

```
Run: PresAnnot x
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Via la base de données :
10.0
Process finished with exit code 0
```

Modifions l'implémentation IDao à utiliser dans @Qualifier de MetierImpl et utilisons la version Web Services. On aura :

```
MetierImpl.java
2
3 import dao.IDao;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.beans.factory.annotation.Qualifier;
6 import org.springframework.stereotype.Component;
7
8 @Component
9 public class MetierImpl implements IMetier{
10     @Autowired
11     @Qualifier("daoVWS")
12     private IDao dao;
13
14     public void setDao(IDao dao) { this.dao = dao; }
15
16
17
18     @Override
19     public double calcul() {
20         double nb = dao.getValue();
21         return 2*nb;
22     }
23 }
```

Quand on exécute, on obtient :



```
Run: PresAnnot x
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Version Web Services :
40.0
Process finished with exit code 0
```

Nous venons ainsi de faire l'injection des dépendances de deux manières différentes en utilisant le framework Spring.